Benchmarking OPeNDAP services for modern ESM data workloads

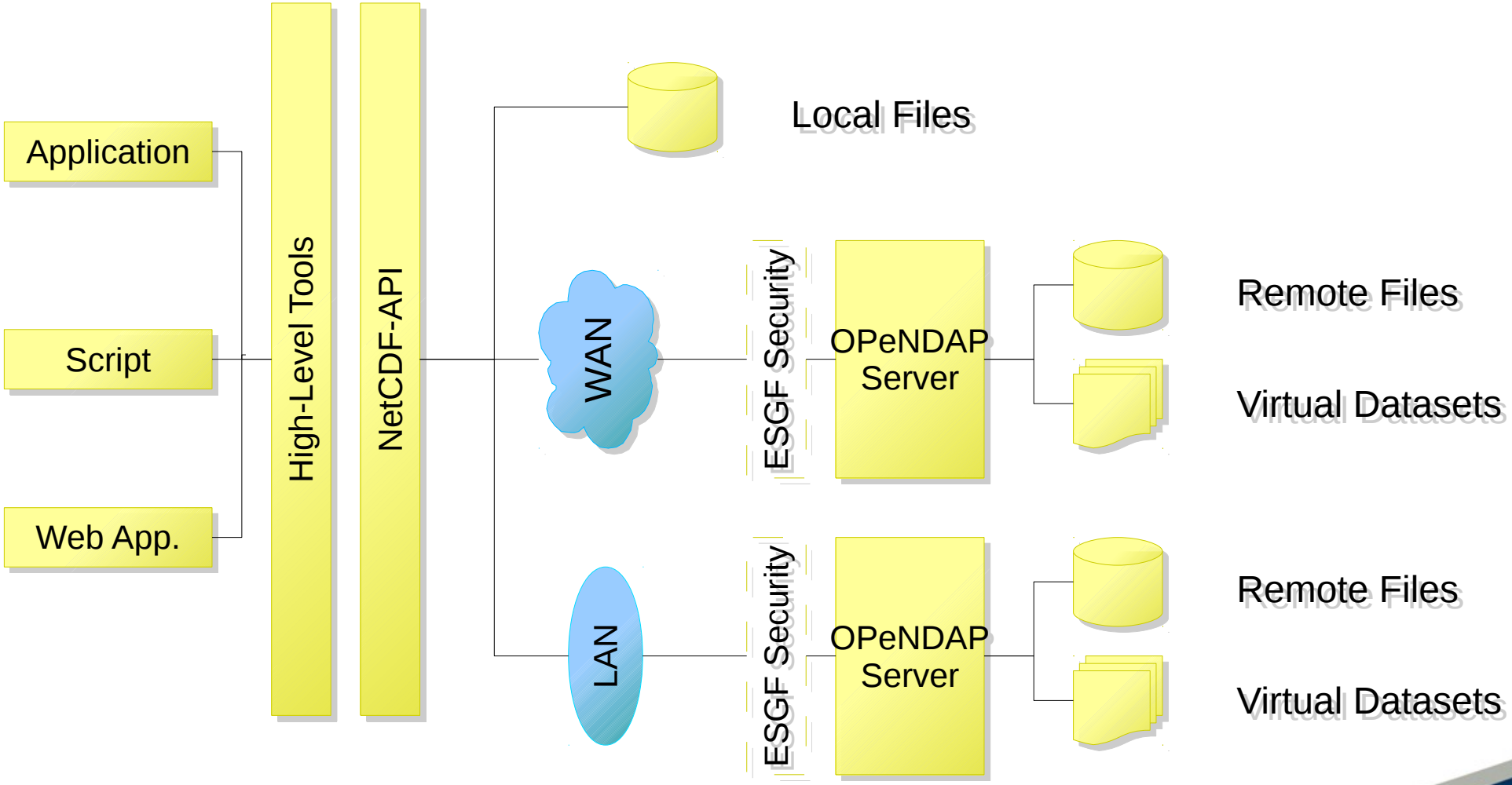
Stephen Pascoe (Stephen.Pascoe@stfc.ac.uk)

Richard Wilkinson (Tessella plc. Abingdon, UK)
Phil Kershaw (Philip.Kershaw@stfc.ac.uk)

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESERACH COUNCIL

BADC:	British Atmospheric Data Centre
NEODC:	Near-Earth Observation Data Centre
IPCC-DDC:	IPCC Data Distribution Centre

Opportunities for the "NetCDF/OPeNDAP Platform"



Key Questions

- What performance can we expect?
- Can OPeNDAP servers cope?
- How can we improve?

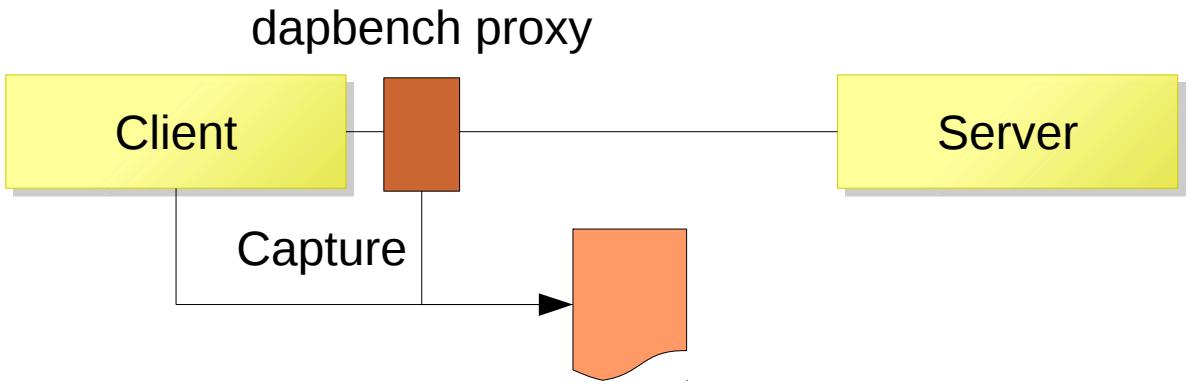
dapbench
OPeNDAP Test Framework

http://github.com/stephenpascoe/dapbench

- Intercept and records OPeNDAP requests from clients
- Load-test OPeNDAP servers
- Jython NetCDF abstraction layer
- Verifying ESG security constraints

- Using: Python, Jython, Grinder, Pandas, Matplotlib, Nmon

Client Tests



Test Clients	Test Dataset
• CDAT (cdat-lite 6.0-alpha-3) • CDO 1.4.7	• 4D temperature field • shape [120,4,144,192] time/level/lat/lon • 51MB NetCDF file.

Test Requests

1. Take a 45ºx45º subset of entire field
2. Regrid entire field to 100x100 lat/lon resolution

Client Results

Test	Tool	DODS Requests	Download Size (Mb)	Comment
Subset	CDO	481	50	1 + time*level requests
	CDAT	17281	1.5	1 + time*level*lat requests
Regrid	CDO	481	50	1 + time*level requests
	CDAT	69121	50	1 + time*level*lat requests

CDO downloads too much and with sub-optimal number of requests. CDAT downloads just enough but with very sub-optimal number of requests. CDAT iterates over all outer dimensions.

ESGF Security testing

```
$ dapbench-thredds <cmd> <options>
```

- Detect OPeNDAP URLs from THREDDS URL or URL list
- Test access to all responses
  - .das, .dods, .dds, .ascii
- Confirms authz redirect correctness
- Useful for C.I. Testing

Server Tests

Test Framework	Test Dataset
• Grinder load-testing framework • NetCDF-Java API • Jython NetCDF abstraction layer • Nmon server monitoring	• 30 files of 4D temperature field • shape [120,4,144,192] time/level/lat/lon • 600MB per NetCDF file.

Test Requests

1. Download a single file (non-dap response)
2. Request all of a random file in **n** slices, **n** = [15, 30, 60, 120, 240, 720, 1440]
3. Continuously reqest random subsets in **m** threads, **m** = [1, .. 40]

Test System

Measured Bandwidth: 930Mbps (iperf)

- DELL optiplex 980.
- Intel i5 4-cores @ 3.2GHz
- 1Gbps NIC
- NetCDF 4.1.2-beta2, HDF5-1.8.4-patch1

- Xen Virtual Machine
- Host: DELL PowerEdge 2950 III
  - 2x quad-core CPU, 24GB RAM
- Guest VM: Paravirtualised OpenSuSE 11.0
  - 2 CPU, 8GB RAM, 4GB swap

Test Servers

- THREDDS Data Server: 3.17.3.1
- Pydap: 3.0.rc.15. netCDF4-python-0.9.3.

Platforms

- 64-bit Java SDK 1.6.0-13, Tomcat-6.0.20. JAVA_OPTS=-Xms1000M -Xmn500M -Xmx1500M
- Apache-2.2.8 (prefork), mod_wsgi-3.3

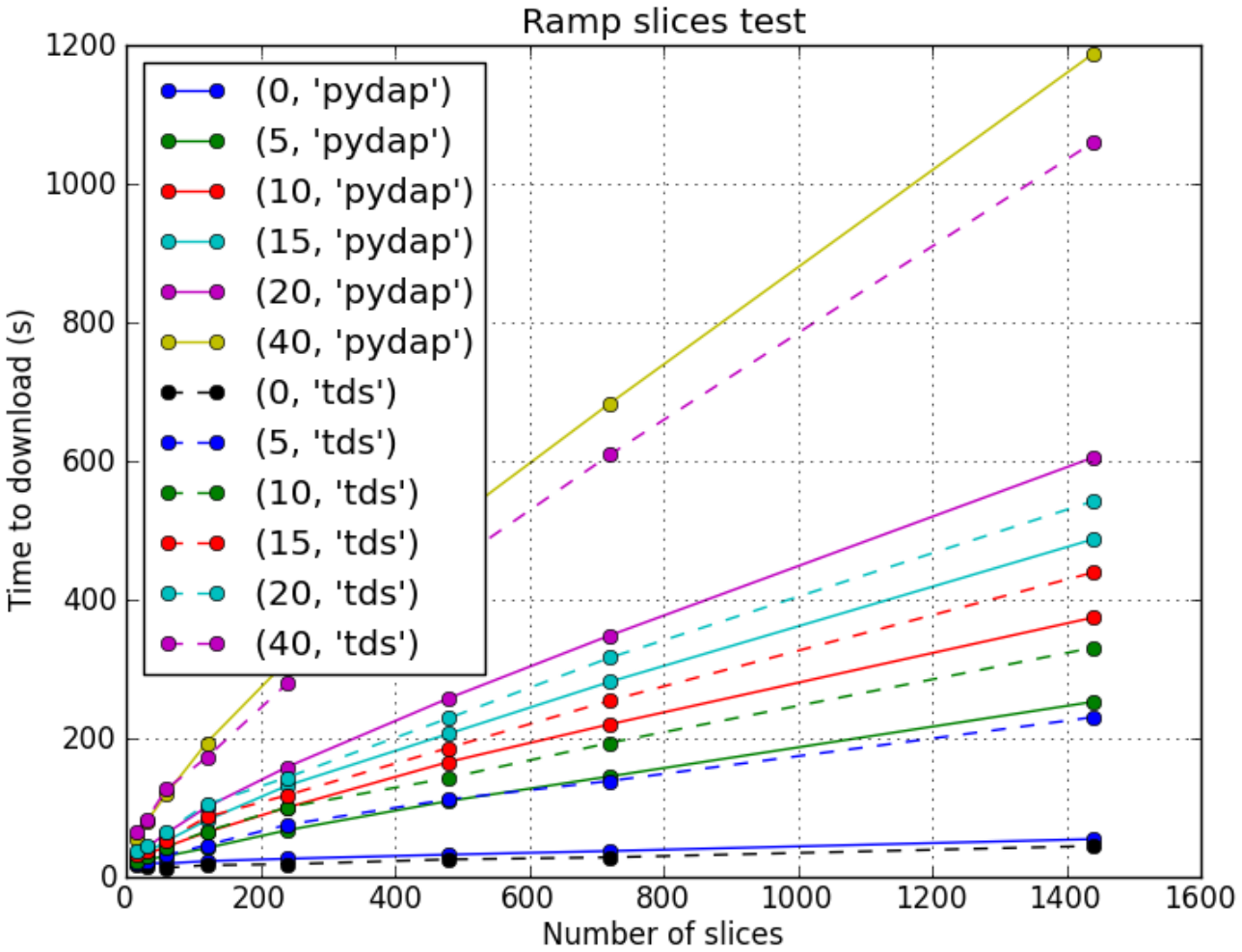
Simulating WAN performance

```
$ tc qdisc add dev eth0 root netem delay <n>ms
```

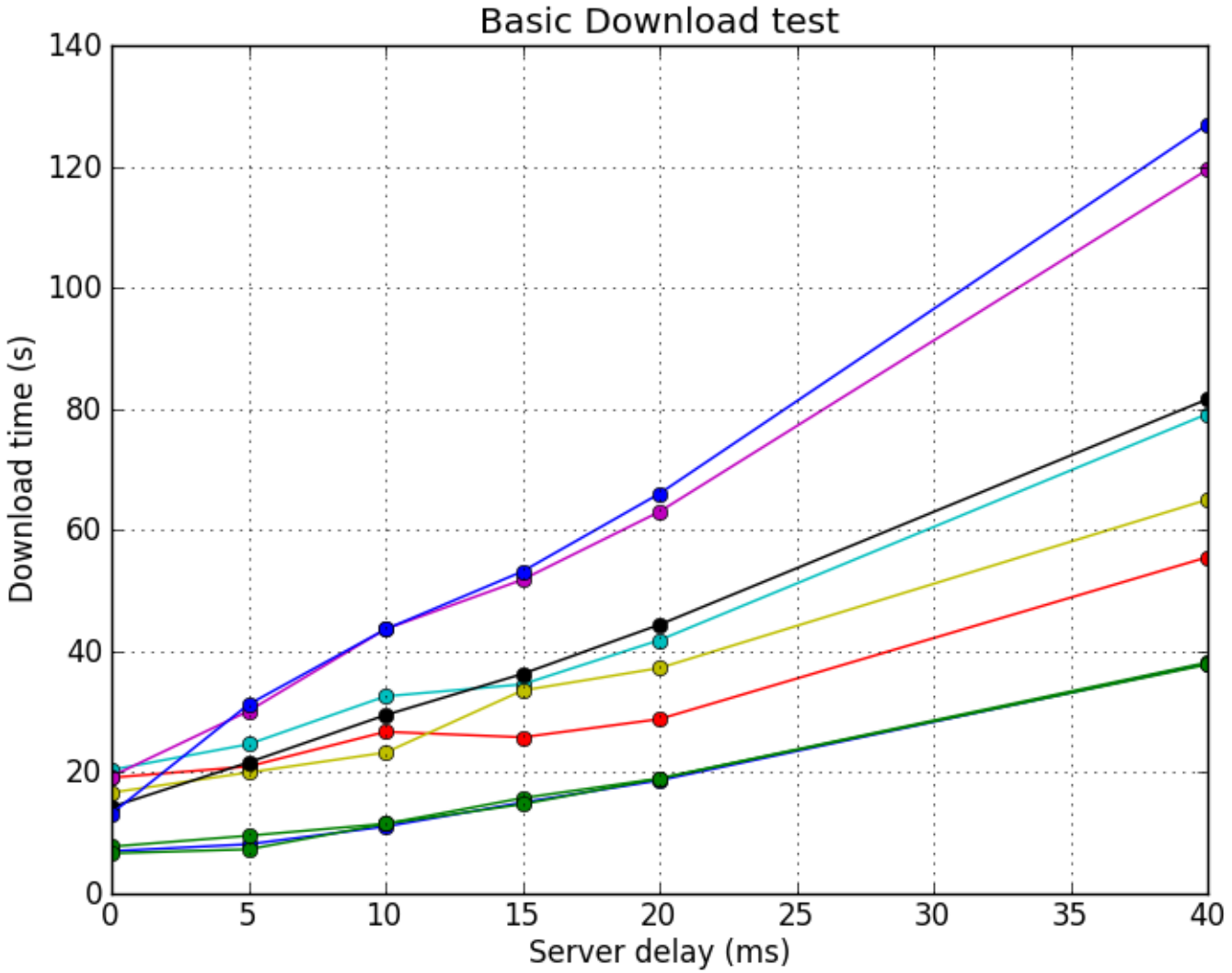
Host	Latency (ms)
CEDA	<0.5
PCMDI	160
DKRZ	40
EGU	>1000

Ha ha ha!

Results: ramp slices



Results: Sequential Download



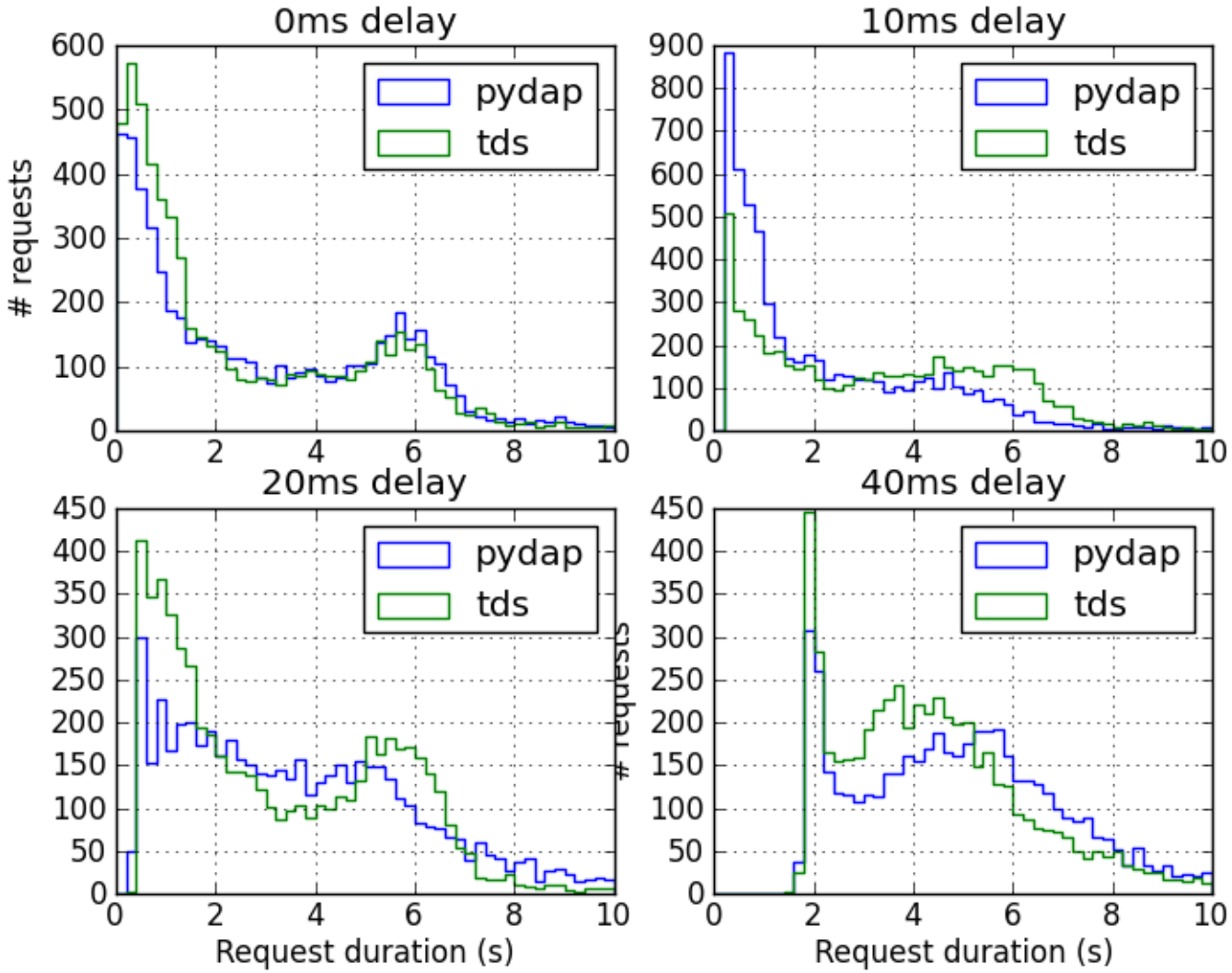
60 slices

30 slices

15 slices

Direct download
Including apache

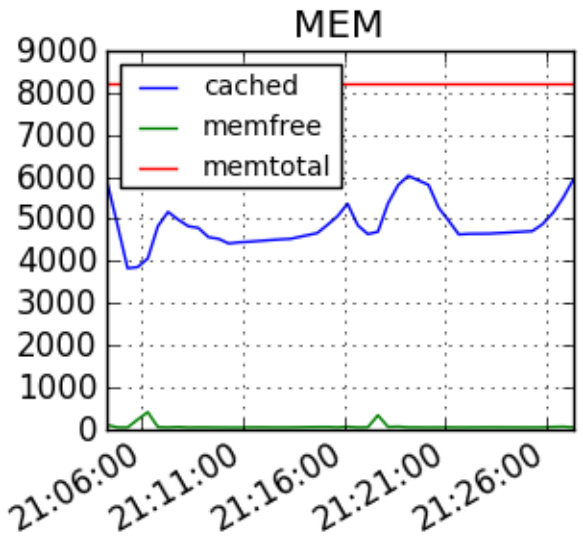
Results: parallel requests

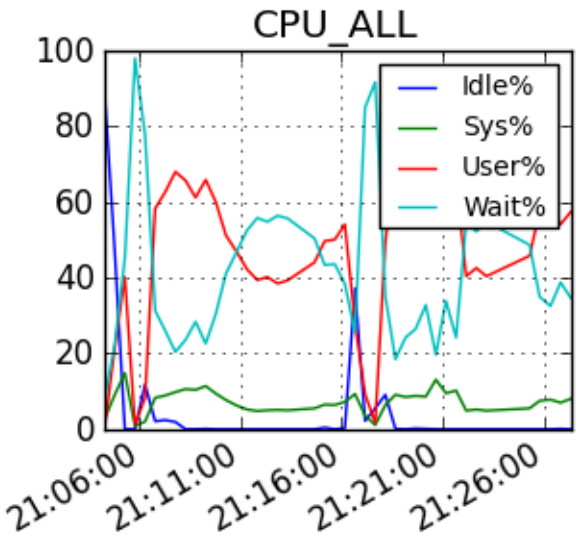


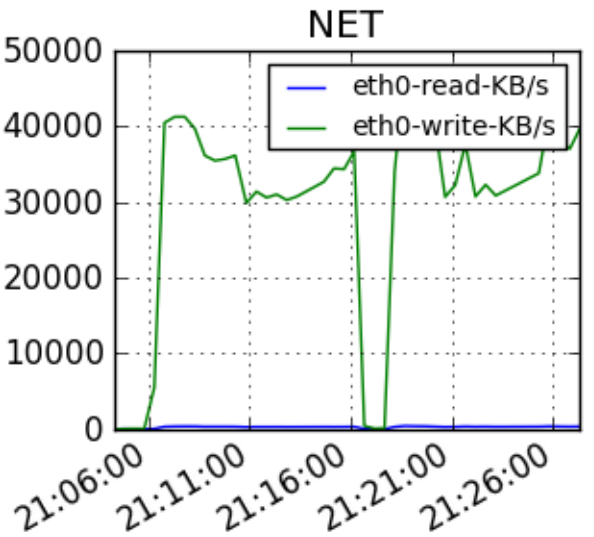
- Select Random File
- Select random time slices
- 4 Processes
- 60 Threads/Process

Results: Server load

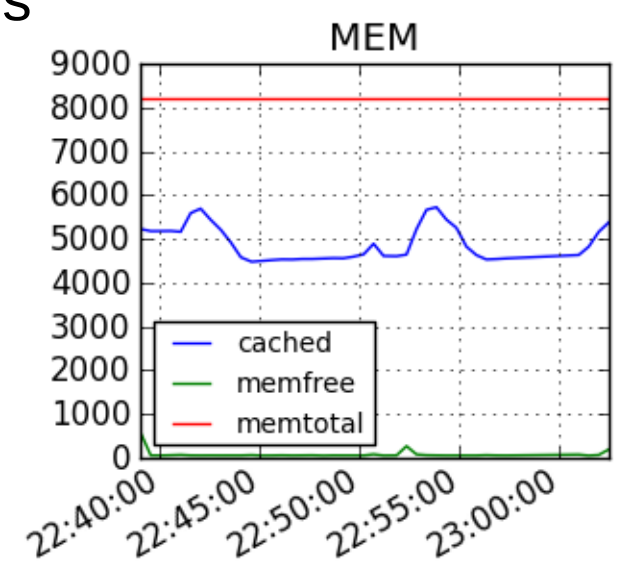
0ms

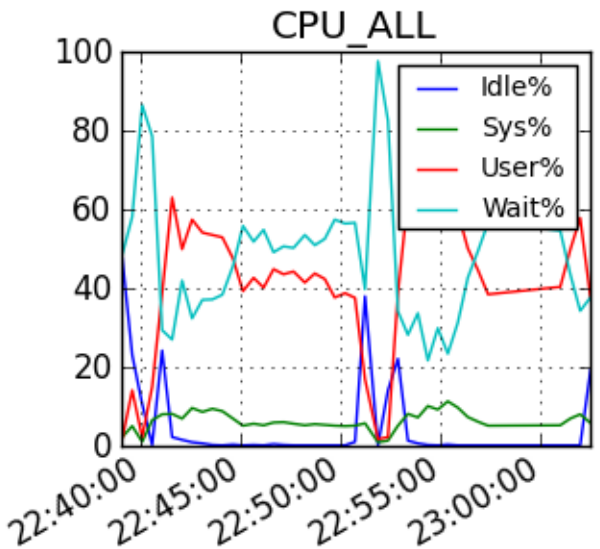


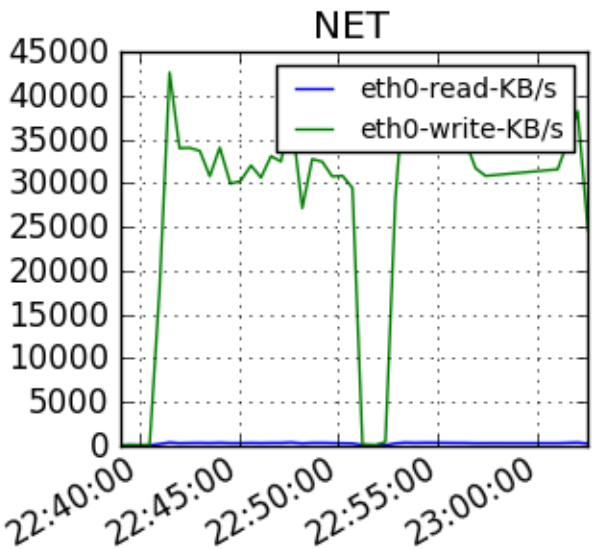




20ms

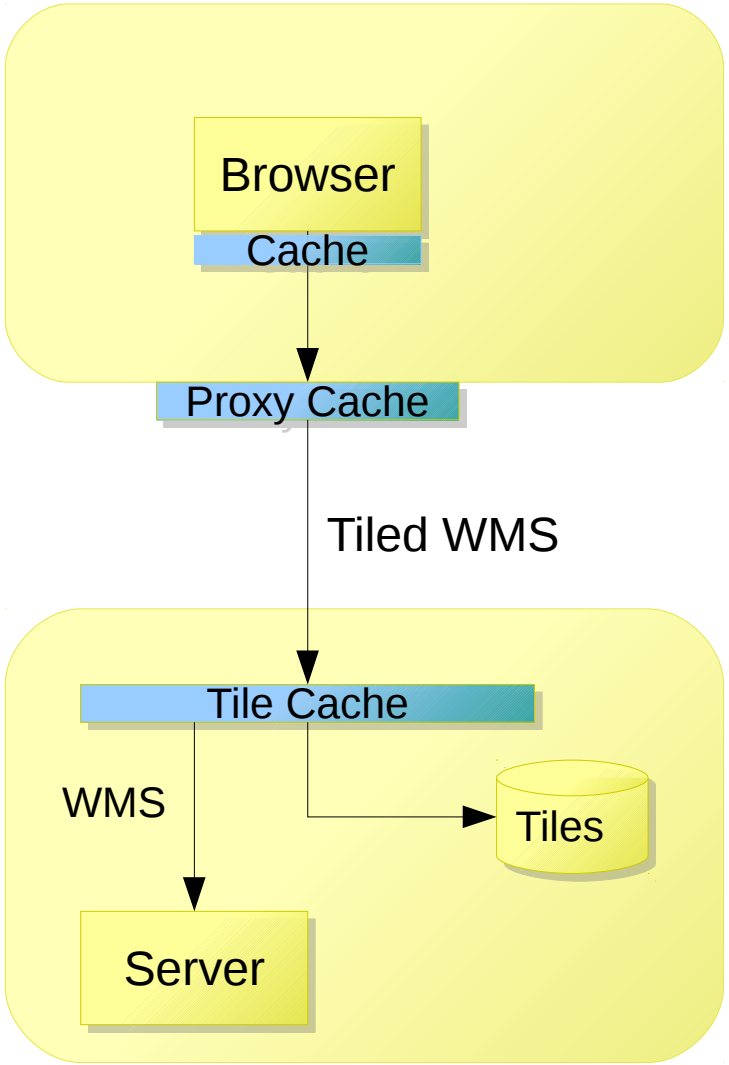




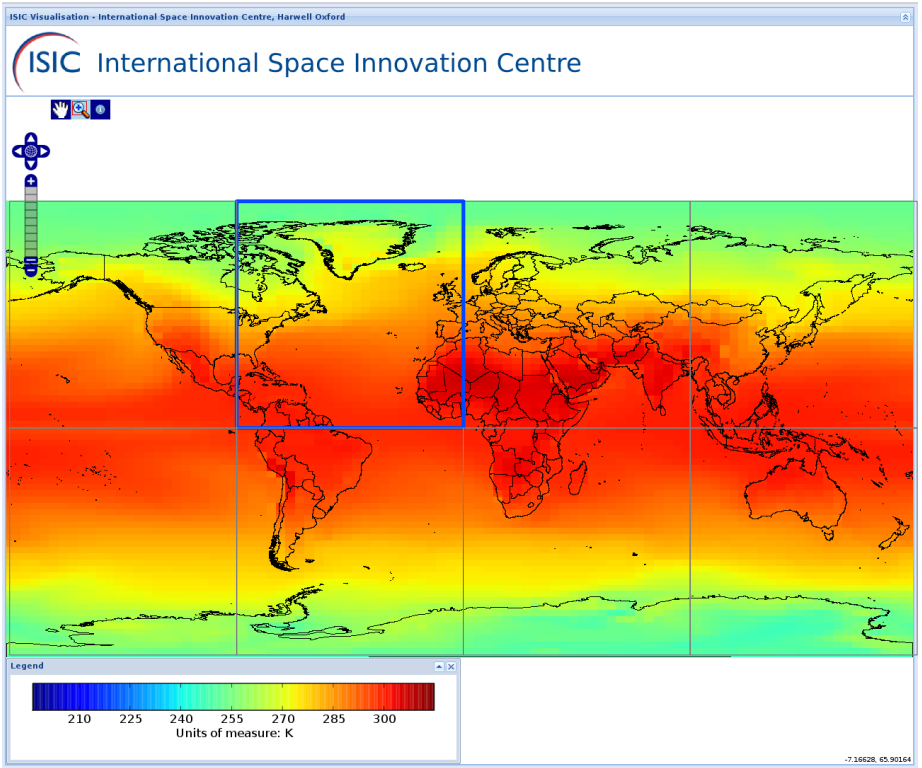


How do we save clients from themselves?

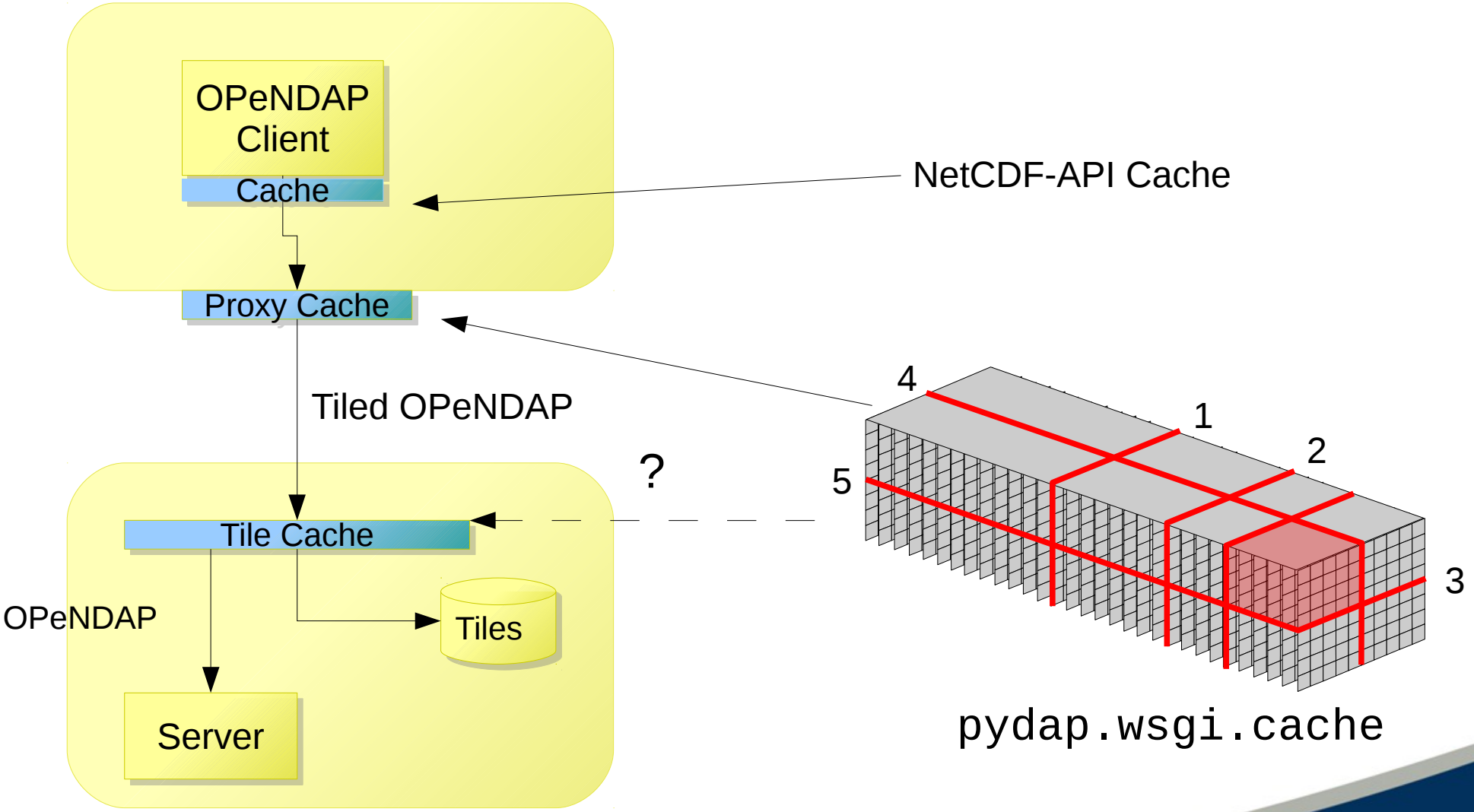
DAP Response tiling / chunking



WMS web-clients use tiling to improve client & server performance



Possible Future



Conclusions

- High-level clients need optimising to use OPeNDAP effectively
  - or we need inteligent proxies / caches
- OPeNDAP servers can compete with whole-file download
  - If you minimise the number of requests
- Server implementations broadly competitive
- Effective caching needs to include clients and/or client-side proxies

Thanks

Stephen.Pascoe@stfc.ac.uk
Twitter: @spascoe
G+: Stephen Pascoe

http://github.com/stephenpascoe/dapbench